TVM @ EdgeCortix - Heterogenous AI Hardware Acceleration

Antonio Tomas
DL Compiler Engineer

www.edgecortix.com
EdgeCortix Overview

- Founded in mid 2019
- Engineering driven with nearly equal mix of ML software, ASIC and FPGA experience

Goal

- ML Acceleration within heterogeneous platforms
- Optimized for low-power edge deployment
- Scalable & modular IP
- **Energy Efficient ML Inference System** rather than yet another processor

“Next generation runtime reconfigurable AI processing architecture for edge inference”
Summary of our Machine Learning Hardware Architecture

Dynamic Neural Accelerator® IP
- Dataflow array architecture
- Runtime-reconfigurable Interconnect
- Heterogenous compute engines
- Scalable ML Inference accelerator

SAKURA AI co-processor
- Upto 52 TOPS (INT8)
- Optimized for batch size 1
- 10W-15W TDP

Form factor
- Dual M.2 and Low-profile PCI-e
- PCI-E gen. 3.0
- LPDDR4x
- JTAG
- UART
The Machine Learning Accelerator Deployment Challenge!

Source of your trained model

- PyTorch
- TensorFlow
- TensorFlow Lite
- ONNX
- mxnet
- Caffe2

Heterogenous compute

- CPUs
- Custom ASIC
- FPGAs

Flexible Compiler Stack

- EdgeCortix SAKURA
- MERA + tvm
High level summary of TVM with EdgeCortix MERA

We embrace three fundamentals parts of TVM to extend our own software stack - MERA

- Native support for Pytorch and TFLite front-ends
- Relay’s QNN dialect for parsing INT8 quantized models at the framework level
- Bring Your Own Code Gen. extension:
  - MergeComposite (patterns)
    - Pattern matching of supported subgraphs (Composites)
  - AnnotateTarget()
  - MergeCompilerRegions()
    - Further merge composites into a single pattern to enable our compiler to have a larger scope for applying further low-level optimizations
  - PartitionGraph()
The MERA Compiler and Software Stack

Pre-optimized

EdgeCortix Model Zoo
- Classification
- Detection
- Segmentation
- Others

Custom Models

Open-source community driven models

Calibration and Quantization

Relay Intermediate Representation (INT8)

Graph Optimization & Partitioning

MERA Code generator

CPU / Host Code Generator

MERA Graph IR

LLVM low-level IR

MERA API

LLVM

Apache TVM Frontend

Runtime

Profiler

Interpreter

Simulator

Dynamic Neural Accelerator

Host CPU

PyTorch

TensorFlow

TensorFlow Lite

ONNX
Evolution of the Intermediate Representations

Model (ONNX/PyTorch/TensorFlow) → Translator/compiler (torch.jit.trace/TFLiteConverter) → Intermediate representations (ONNX/TorchScript/TFLite format)

Conversion to TVM RelayIR with QNN dialect → TVM Frontend → TVM Relay IR

MERA optimized pattern matching, BYOC flow and Accelerator specific optimizations → MERA Code generator → Optimized MERA IR

AI accelerator consumes MERA IR and emits binary artifacts → Binary artifacts

ONNX PyTorch TensorFlow

TVM

MERA Compiler

EdgeCortex DNA IP
Flexible Graph Partitioning

Relay

Partitioning

Merge compiler regions

Compilation and execution

LLVM

ARM/X86

1 output

MERA

DNA Runtime

1 input

DNA IP

1 output

DDR Memory

ARM or X86

2 inputs

2 outputs
MERA API to Deploy any Pre-trained Model / Models

1. Framework level Quantization supported

   - PyTorch
   - TensorFlow Lite
   - ONNX

   Pre-trained FP32 Model → Calibration → Quantization to INT8

   Take standard machine learning framework dependent steps to calibrate and quantize the model.

2. Specify input size eg. (1920x1080)
   Build quantized graph with MERA API *
   Choose Target
   - Profiler
   - DNA IP on FPGA
   - EdgeCortix AI Chip

3. Heterogenous platforms
   Generates deploy directory with compiled artifacts
   - EdgeCortix Sakura AI co-processor
   - 3rd party FPGAs with EdgeCortix DNA IP

   * Supports both x86 and Arm based host architectures, enabling execution on SoC or PCI-E type cards
Challenges & Observations of using TVM with a custom AI accelerator

• Software is the most critical element for efficient deployment with any hardware platform.

• A general deep learning compiler like TVM (at a high level) may help achieve a common standard across vendor specific software tools - better user experience & reduced adoption time

However, there are few key challenges (missing features) that we observed needs more attention from the open-source community. These are also some areas we are actively working on.

• Autonomous and efficient resource scheduling & memory allocation
• Efficiently deploying multiple models
• Runtime pipelining between processor and co-processors
• Built in quantization support for mixed precision networks
• Bring your own sparse networks / pruning techniques